# Energy Based Models, Restricted Boltzmann Machines and Deep Networks

Jesse Eickholt

# ???? 

Who’s heard of …

- Energy Based Models (EBMs)
- Restricted Boltzmann Machines (RBMs)
- Deep Belief Networks
- Auto-encoders

# Objectives

1. Awareness of new developments in statistical machine learning
2. Exposure to Energy Based Models, RBMs and Deep Belief Networks
3. Generate some excitement about these new developments

# Outline

- Motivating factors for study
- RBMs
- Deep Belief Networks
- Applications

# The Toolbox

We often reach for the familiar…

For discriminative tasks we have

- o neural networks (~1980’s, back-prop)
- o SVM (~1990’s, Vapnik)

But is there anything better out there???

# Challenges with SVM/NN

<u>Potential difficulties with SVM</u>

- Training time for large datasets
- Large number of support vectors for hard classification problems

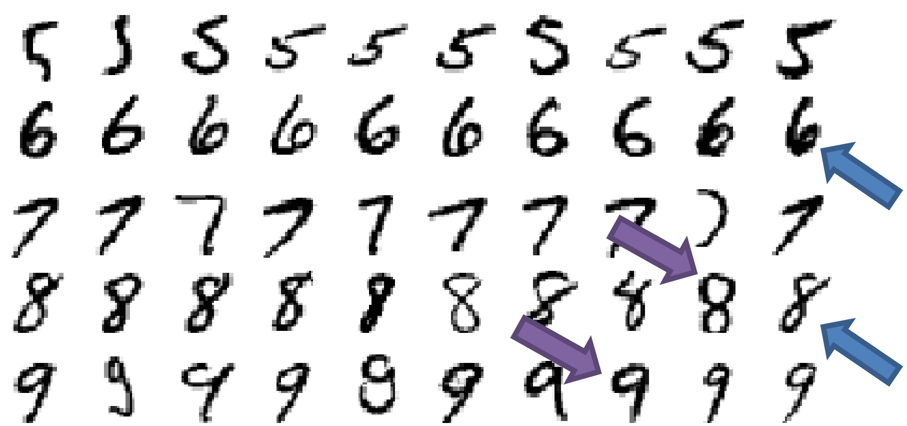
<u>Potential difficulties with NN & back-prop</u>

- Diminishing gradient inhibits multiple layers
- Can get stuck in local minimums
- Training time can be extensive

# Challenges with SVM/NN

More general "problems" with NNs and SVM…

- Need labeled data (what about unlabeled data?)
- Amount of information restricted by labels (ie, hard to learn a complex model if we are limited by labels)



What if I could use "8"s to learn to recognize "6"s ?

# How to respond to these challenges

- Try to model the structure of the sensory input (ie, data), but keep the efficiency and simplicity of a gradient method
  - Adjust the weights to maximize the probability that a generative model would have produced the sensory input.
  - Learn p(data) not p(label | data)
- So instead of learning a label, first learn how to generative your data

Hinton, 2007

# How to respond to these challenges

- Try to model the structure of the sensory input (ie, data), but keep the efficiency and simplicity of a gradient method
  - Adjust the weights to maximize the probability that a generative model would have produced the sensory input.
  - Learn p(data) not p(label | data)
- So instead of learning label, first learn how to generative your data

Immediate benefit in that all data does not have to be label. Also reduces dependency on label.

Hinton, 2007

# Recap

So, we are convinced we …

1. recognize some concerns with “standard” tools and would like what other options are out there
2. like the idea of modeling the input first (ie, building a model of our data as oppose to an out right classifier)

# Energy Based Models

p(x) – probability of our data; data is represented by feature vector **x**.

$$p(x) = \frac{e^{-E(x)}}{Z}.$$

and

$$Z = \sum_x e^{-E(x)}$$

Attach an energy function (ie, *E*(x)) to score a configuration (ie, each possible input x).

We want desirable data to have low energy. Thus, tweak the parameters of *E*(x) accordingly.

*Restricted Boltzann Machines (RBM)*

# EBMs with Hidden Units

To increase power of EBMs, add hidden variables.

$$P(x) = \sum_h P(x,h) = \sum_h \frac{e^{-E(x,h)}}{Z}.$$

By using the notation,

Free energy

$$\mathcal{F}(x) = -\log \sum_h e^{-E(x,h)}$$

We can rewrite p(x) in a form similar to the standard EBM,

$$P(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \text{ with } Z = \sum_x e^{-\mathcal{F}(x)}.$$

*Restricted Boltzmann Machines (RBM)*

# Tweakin' Parameters

Now we need to adjust the model so it reflects our data, do ML

- Likelihood fn

$$\mathrm{L}(\theta) = \Pi_{i=1}^{n} p(x_i;\theta)$$

- Avg. Log-likelihood fn

$$\ell(\theta) = \frac{1}{n}\log(\Pi_i p(x_i;\theta)) = \frac{1}{n}\sum_i \log(p(x_i;\theta))$$

$$= \frac{1}{n}\sum_i log\frac{e^{-F(x_i)}}{Z} = \frac{1}{n}\sum_i(-F(x_i) - \log(Z))$$

# Tweakin' Parameters

- Take the derivative

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \frac{1}{n}\sum_i (\frac{-\partial F(x_i)}{\partial \theta_j} - \frac{\partial \log Z}{\partial \theta_j}) = \frac{1}{n}\sum_i (\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\frac{\partial Z}{\partial \theta_j})$$

$$= \frac{1}{n}\sum_i (\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\sum_{\hat{x}} e^{-F(\hat{x})}\frac{\partial F(\hat{x})}{\partial \theta_j})$$

$$= \frac{1}{n}\sum_i (\frac{-\partial F(x_i)}{\partial \theta_j}) + \sum_{\hat{x}} p(\hat{x})\frac{\partial F(\hat{x})}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_i (\frac{-\partial F(x_i)}{\partial \theta_j}) + E_p[\frac{\partial F(x)}{\partial \theta_j}]$$

# Tweakin' Parameters

- Take the derivative

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} - \frac{\partial \log Z}{\partial \theta_j}\right) = \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\frac{\partial Z}{\partial \theta_j}\right)$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\sum_{\hat{x}} e^{-F(\hat{x})}\frac{\partial F(\hat{x})}{\partial \theta_j}\right)$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j}\right) + \sum_{\hat{x}} p(\hat{x})\frac{\partial F(\hat{x})}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j}\right) + E_p\left[\frac{\partial F(x)}{\partial \theta_j}\right]$$

Can think of as an expectation over dataset.

This is an expectation over all possible configurations of input x. ?#@! Grows exponentially as function of the length of input

*Restricted Boltzann Machines (RBM)*

# Transition to RBM

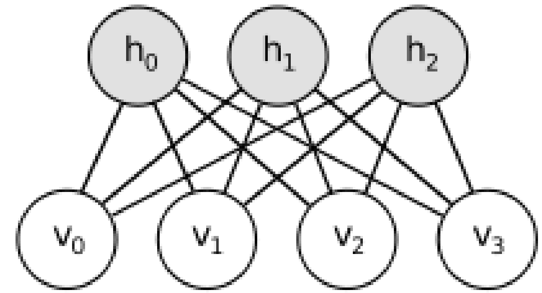
Looks like training a EBM is, in general, a tall task. But after much

Jump to an end result…

Restricted Boltzmann Machines (RBM)

# RBMs

- Represented by a bipartite graph, with symmetric, weighted connections
- One layer has visible nodes and the other hidden (ie, latent) variables.
- Notes are often binary , stochastic units (ie, assume 0 or 1 based on probablity)



# The Energy Function

$$E(v,h) = -b'v - c'h - h'Wv$$

Remember that v and h are vectors of binary units; b, c and W are real number valued

Or, ignoring the bias terms

binary state of visible unit i

binary state of hidden unit j

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

weight between units i and j

Slide modified from Hinton, 2007

# What's gained by "Restricted"

1) Conditional probabilities factor nicely

$P(h|v) = \Pi_i P(h_i|v)$ and $P(v|h) = \Pi_i P(v_i|h)$

2) Using binary units, we also can get

$$P(v_j = 1|h) = \sigma(b_j + W_j^{'} h)$$
$$P(h_i = 1|v) = \sigma(c_i + W_j v)$$

So we can get a sample of the visible or hidden nodes easily...

*Restricted Boltzann Machines (RBM)*

# Gradient revisited

$$\frac{\partial \log p(x)}{\partial \theta} = \frac{\partial}{\partial \theta}\log(\frac{e^{-F(x)}}{Z}) = \frac{\partial - F(x)}{\partial \theta} - \frac{\partial log Z}{\partial \theta}$$

$$= \frac{\partial log \sum_h e^{-E(x,h)}}{\partial \theta} - \frac{\partial \log \sum_{\hat{x}} \sum_h e^{-E(\hat{x},h)}}{\partial \theta}$$

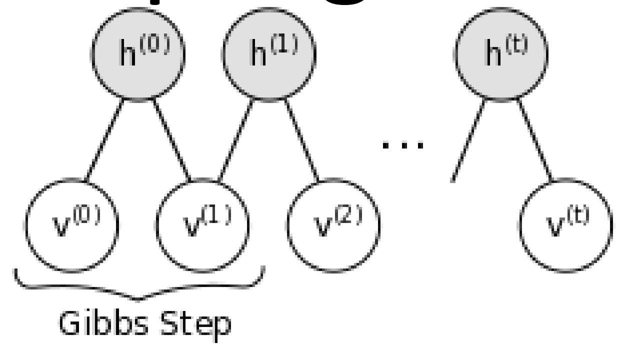
$$= -\frac{1}{\sum_h e^{-E(x,h)}} \sum_h e^{-E(x,h)} \frac{\partial E(x,h)}{\partial \theta}$$

$$+ \frac{1}{\sum_{\hat{x},h} e^{-E(\hat{x},h)}} \sum_{\hat{x},h} e^{-E(\hat{x},h)} \frac{\partial E(\hat{x},h)}{\partial \theta}$$

$$= -\sum_h p(h|x) \frac{\partial E(x,h)}{\partial \theta} + \sum_{\hat{x},h} p(\hat{x},h) \frac{\partial E(\hat{x},h)}{\partial \theta}$$

The first term we can calculate directly from data and we sample from p(v,h) using Gibbs Sampling. [ Remember that x represents the observable variables, *ie* v in RBM ]

# Gibbs Sampling




Can sample from p(v,h) by repeatedly sampling from v and h using the eqns. for p(v|h) and p(h|v).

As $t \rightarrow \infty$, ($v^{(t)}$ ,$h^{(t)}$) converge to samples of p(v,h).

But… hard to know when equilibrium has been reach, can be computionaly expensive

# Learning Rule

Recall energy function

$$E(v,h) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_{i,j} v_i h_j w_{i,j}$$

Calculating derivatives…

$$\frac{\partial E(v,h)}{\partial w_{i,j}} = v_i h_j$$

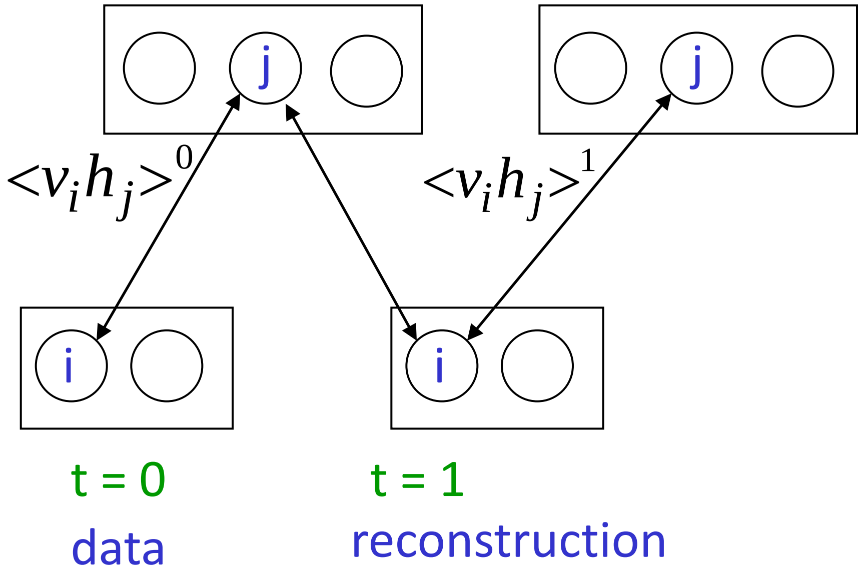
$$\frac{\partial E(v,h)}{\partial b_i} = v_i$$

$$\frac{\partial E(v,h)}{\partial c_j} = h_j$$

So,

$$\Delta w_{i,j} \propto \epsilon(< v_i h_j >^0 - < v_i h_j >^\infty)$$

# A quick way to learn an RBM




Start with a training vector on the visible units.

Update all the hidden units in parallel

Update the all the visible units in parallel to get a "reconstruction".

Update the hidden units again.

$$\Delta w_{ij} = \varepsilon \, ( <v_i h_j>^0 - <v_i h_j>^1 )$$

**This is not following the gradient of the log likelihood**. But it works well. It is approximately following the gradient of another objective function (Carreira-Perpinan & Hinton, 2005).

Slide modified from Hinton, 2007

# Challenges with RBMs

A number of choices to be made

- Types of nodes, learning weight, initial values, batch sizes, etc.
- Care should be taken to avoid over-fitting
- Lack of ready to go software packages

A RBM "manual" is available on line…

http://www.cs.utoronto.ca/~hinton/absps/guideTR.pdf

# Why ???

Okay, we can model p(x).

But how to…

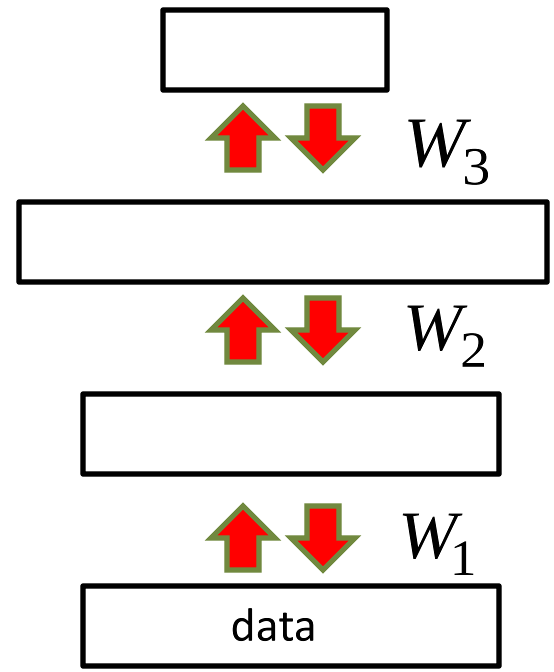
1. **Find p(label|x). We want a classifier!**
2. Improve the model for p(x).

# Deep Belief Nets

RBMs are typically used in stack

- Train them up one layer at a time
- Hidden units become visible units to the next layer up

**If your goal is a discriminator, you train a classifier on the top level representation of your input.**



# Why stack them up? Why does this work?

This is a good question, with a long complicated answer.

Basically, doing so can improve a lower variation bound on the probability of training data under the model.

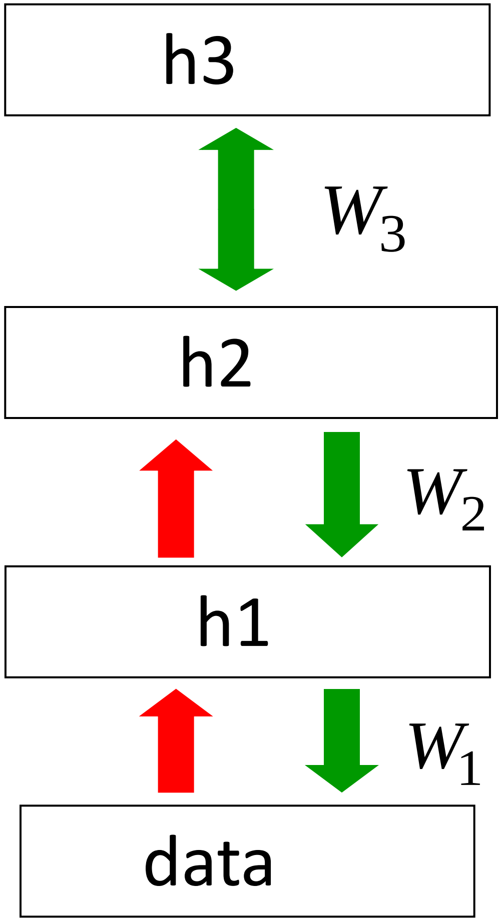
Hinton, Osindero, & The, 2006

# How to generate from the model

- To generate data:
  - Get an equilibrium sample from the top-level RBM by performing alternating Gibbs sampling for a long time.
  - Perform a top-down pass to get states for all the other layers.

So the lower level bottom-up connections are not part of the generative model. They are just used for inference.

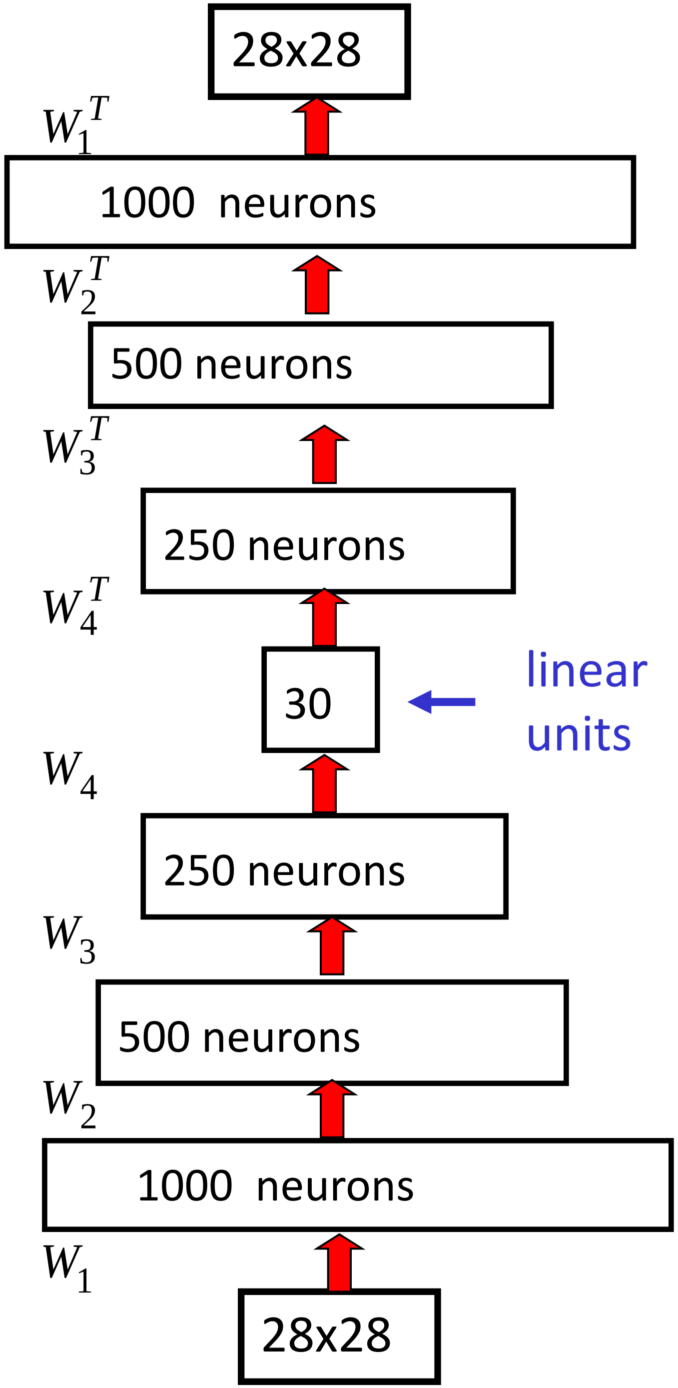
Bonus when modeling p(x), we can see what the model believes in




Slide modified from Hinton, 2007

# Deep Autoencoders

- They always looked like a really nice way to do non-linear dimensionality reduction:
  - But it is very difficult to optimize deep autoencoders using backpropagation.
- We now have a much better way to optimize them:
  - First train a stack of 4 RBM's
  - Then "unroll" them.
  - Then fine-tune with backprop.




Hinton & Salakhutdinov, 2006; slide form Hinton UCL tutorial

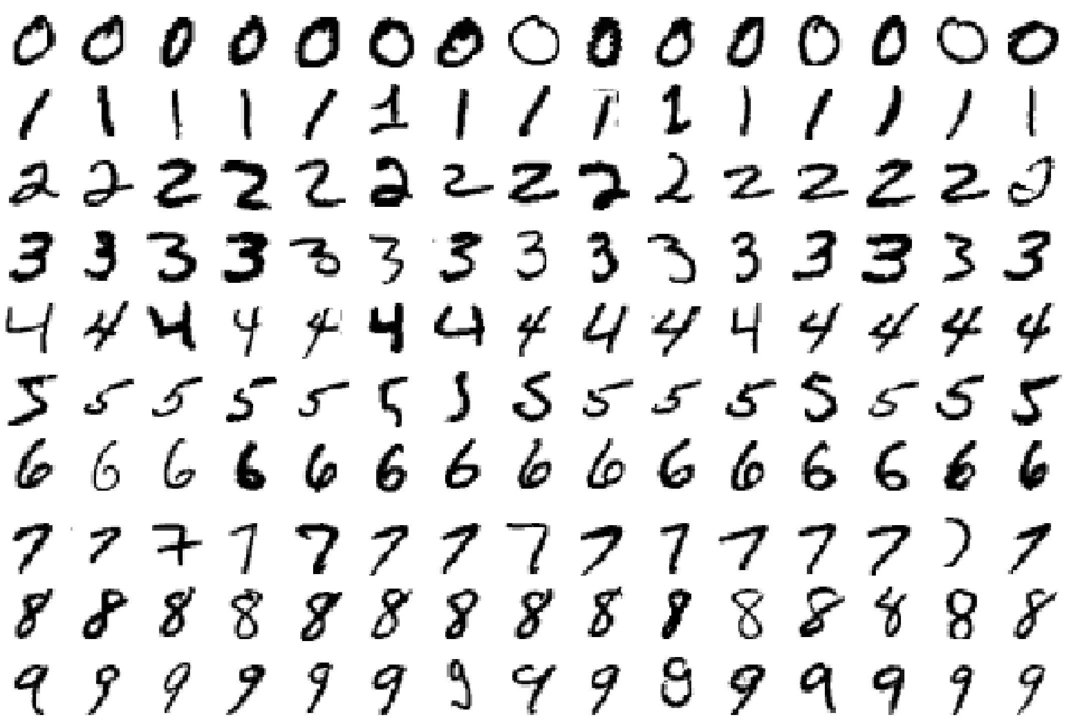
# Some Applications

We will look at two applications done by Hinton's Lab

- A model for digit recognition
- Cluster/search documents

# Applications: A model of digit recognition

- Classify digits (0 – 9)
- Input is a 28x28 image from MNIST (training 60k, test 10k examples)

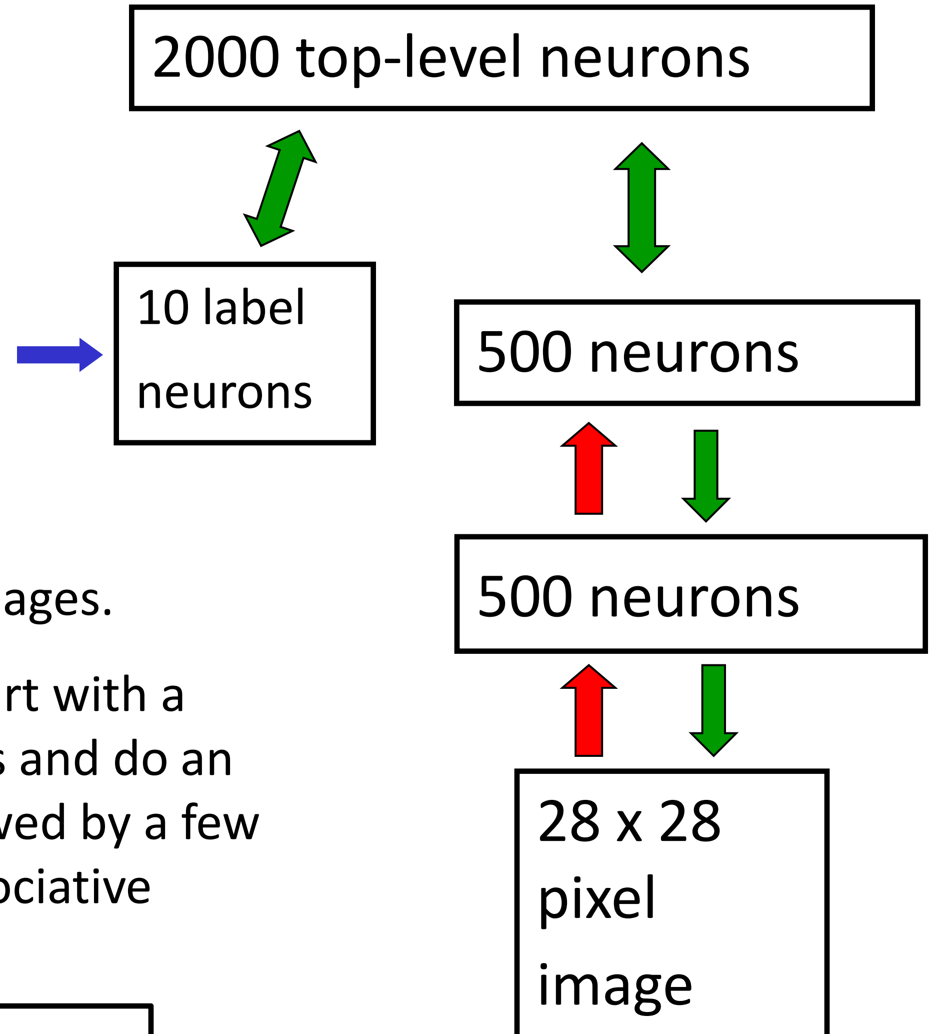


# Applications: A model of digit recognition

This is work from Hinton et al., 2006

The top two layers form an associative memory whose energy landscape models the low dimensional manifolds of the digits.

The energy valleys have names




The model learns to generate combinations of labels and images.

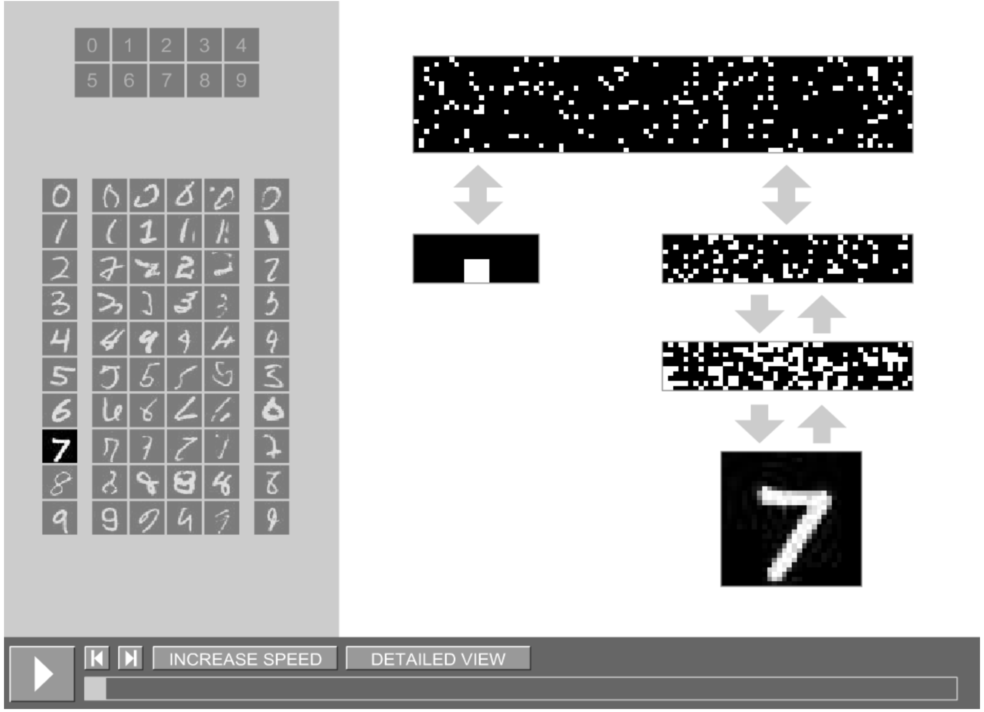
To perform recognition we start with a neutral state of the label units and do an up-pass from the image followed by a few iterations of the top-level associative memory.

Matlab/Octave code available at http://www.cs.utoronto.ca/~hinton/

Slide modified from Hinton, 2007

# Model in action

Hinton has provided an excellent way to view the model in action…




http://www.cs.toronto.edu/~hinton/digits.html

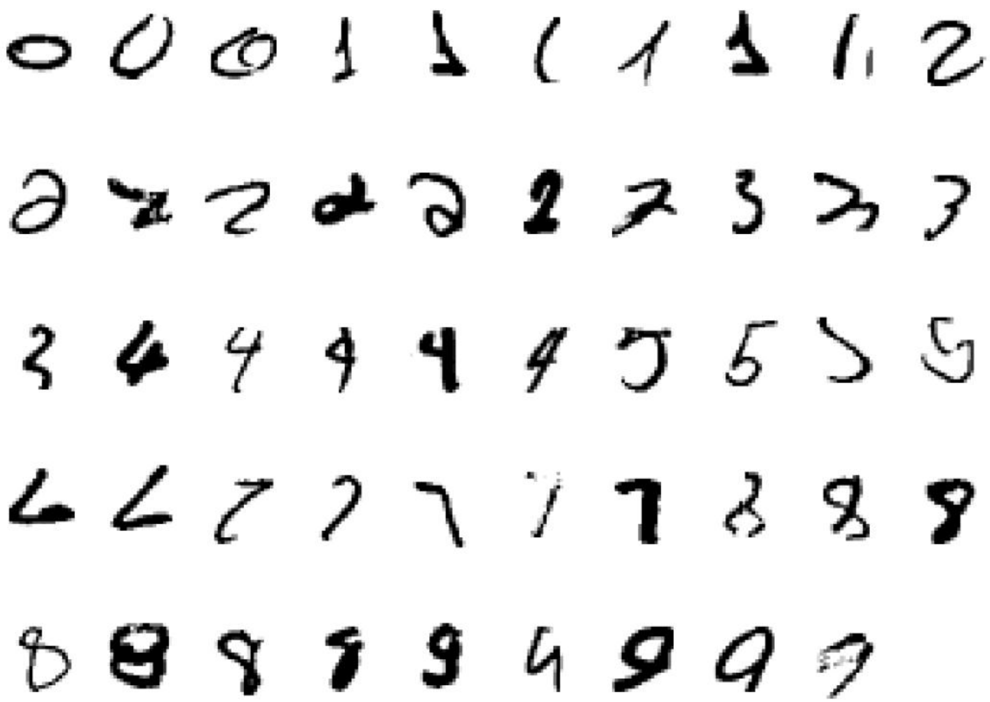
# More Digits

Samples generated by letting the associative memory run with one label clamped. There are 1000 iterations of alternating Gibbs sampling between samples.

Slide from Hinton, 2007

# Even More Digits

Examples of correctly recognized handwritten digits that the neural network had never seen before



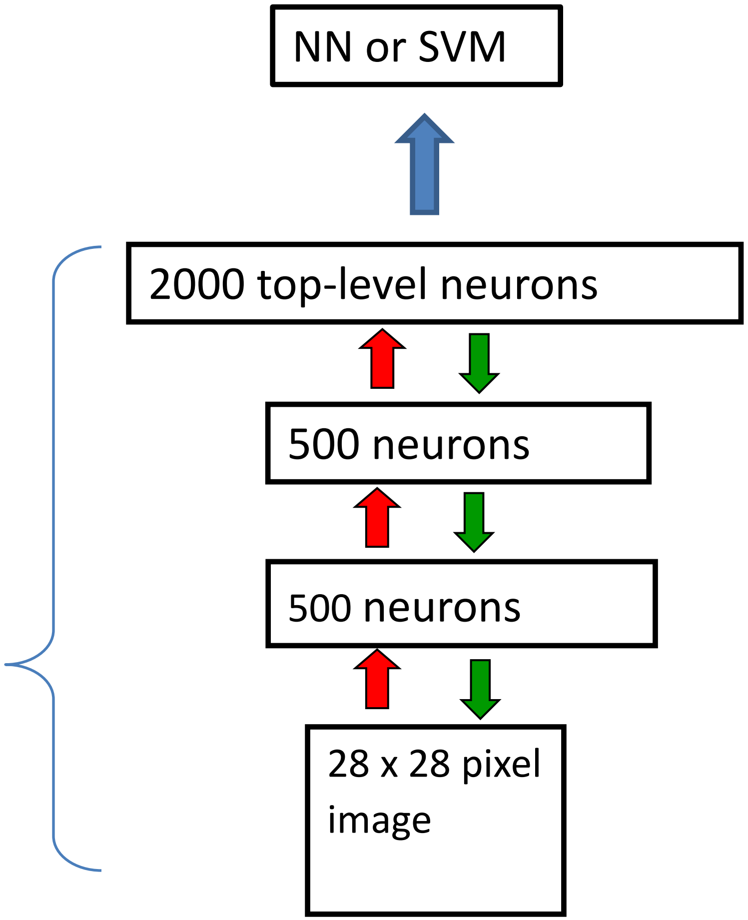
Slide from Hinton, 2007

# Extensions

Do classification.

One way (probably no the best), train generative model with labeled/unlabeled data

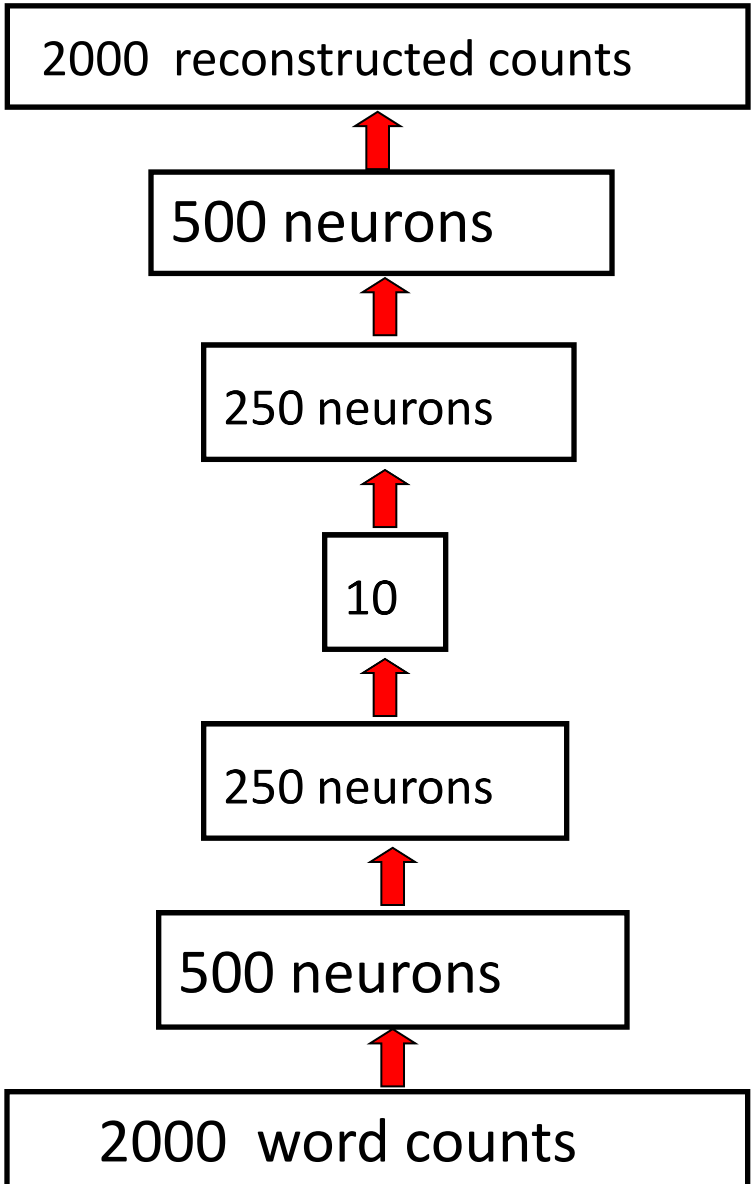
Then train a NN on higher dimensional representation.



# Applications: Classifying text documents

- A document can be characterized by the frequency of words that appear (ie, word counts for some dictionary become feature vector)
- Goals…
  1. Group/cluster similar documents
  2. Find similar documents

# How to compress the count vector

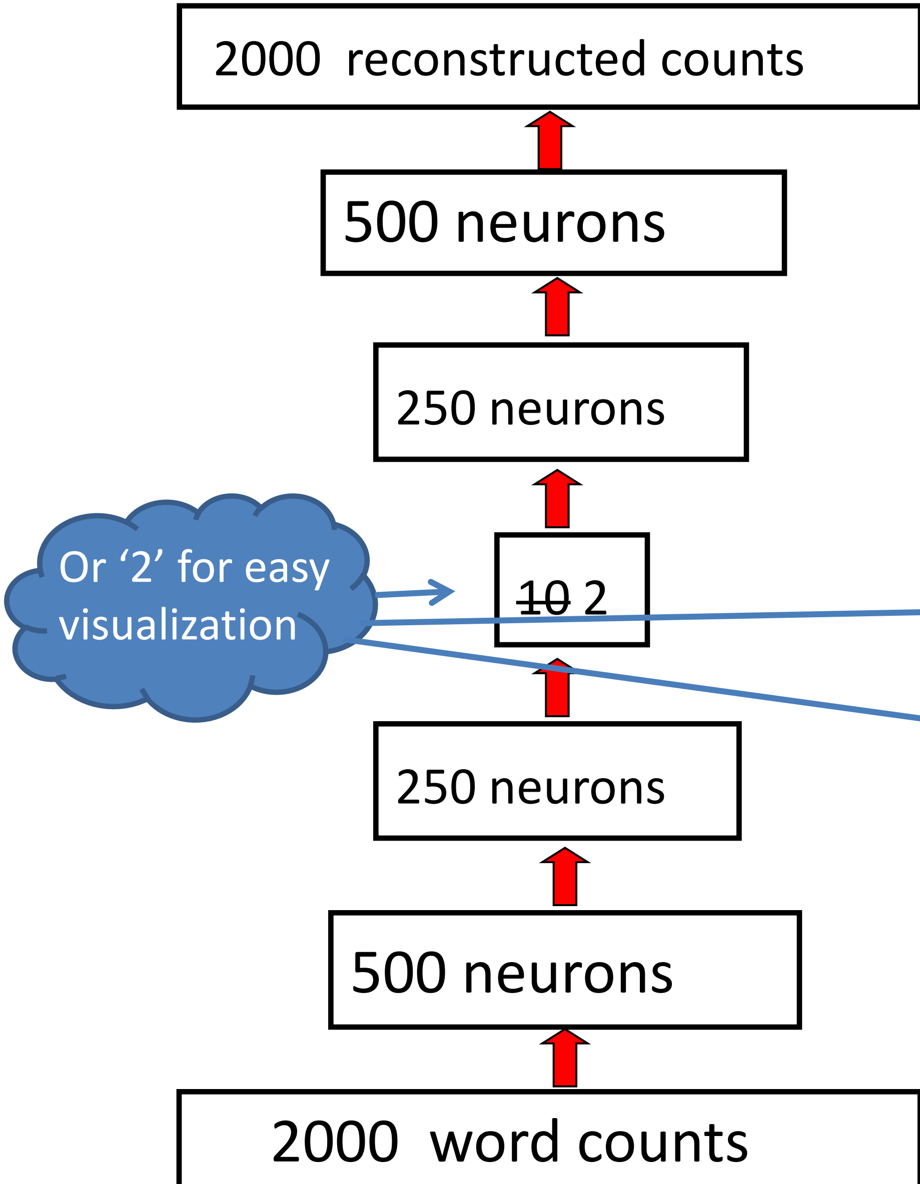



<u>Multi-layer auto-encoder</u>

- Train a model to reproduce its input vector as its output
- This setup forces as much information as possible be compressed and passed thru the 10 numbers in the central bottleneck.
- These 10 numbers are then a good way to compare documents.

Slide modified from Hinton, 2007

# How to compress the count vector

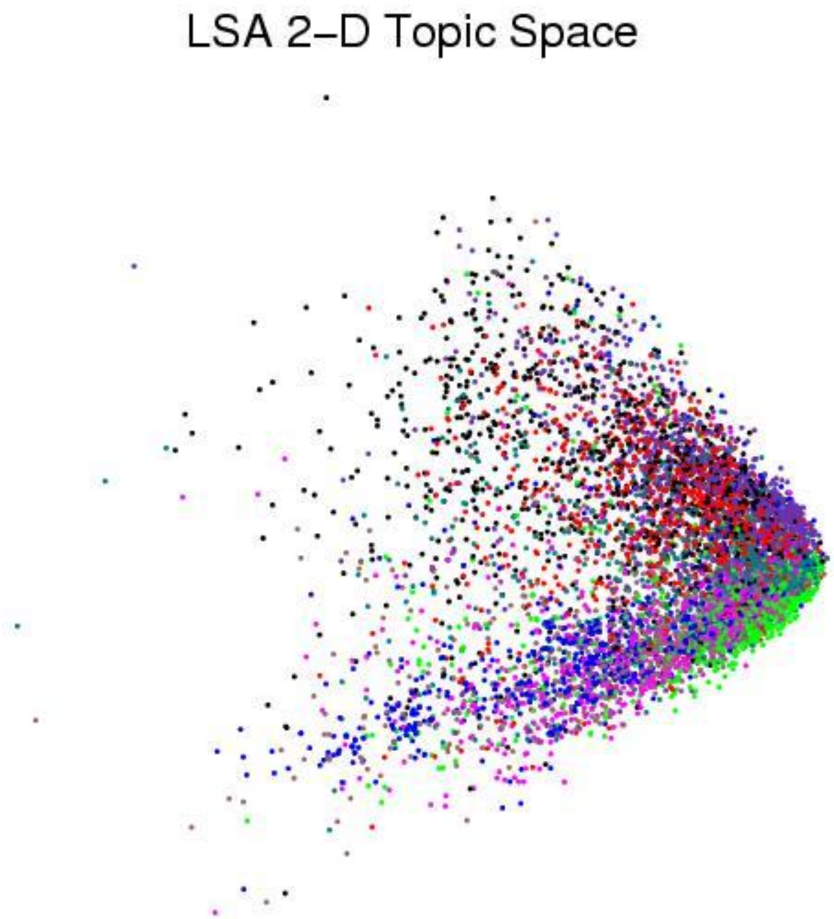


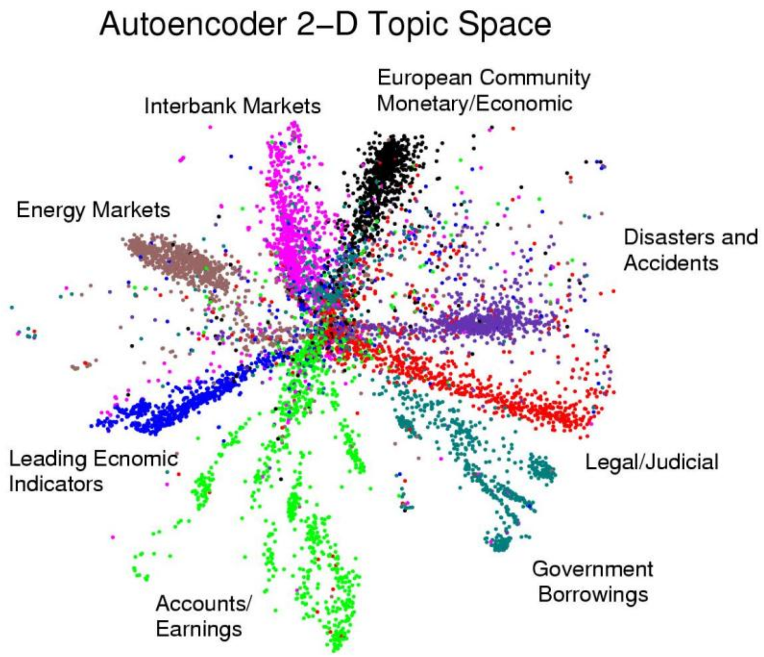

## Multi-layer auto-encoder

- Train a model to reproduce its input vector as its output
- This setup forces as much information as possible be compressed and passed thru the ~~10~~ 2 numbers in the central bottleneck.
- These ~~10~~ 2 numbers are then a good way to compare documents.

Slide modified from Hinton, 2007

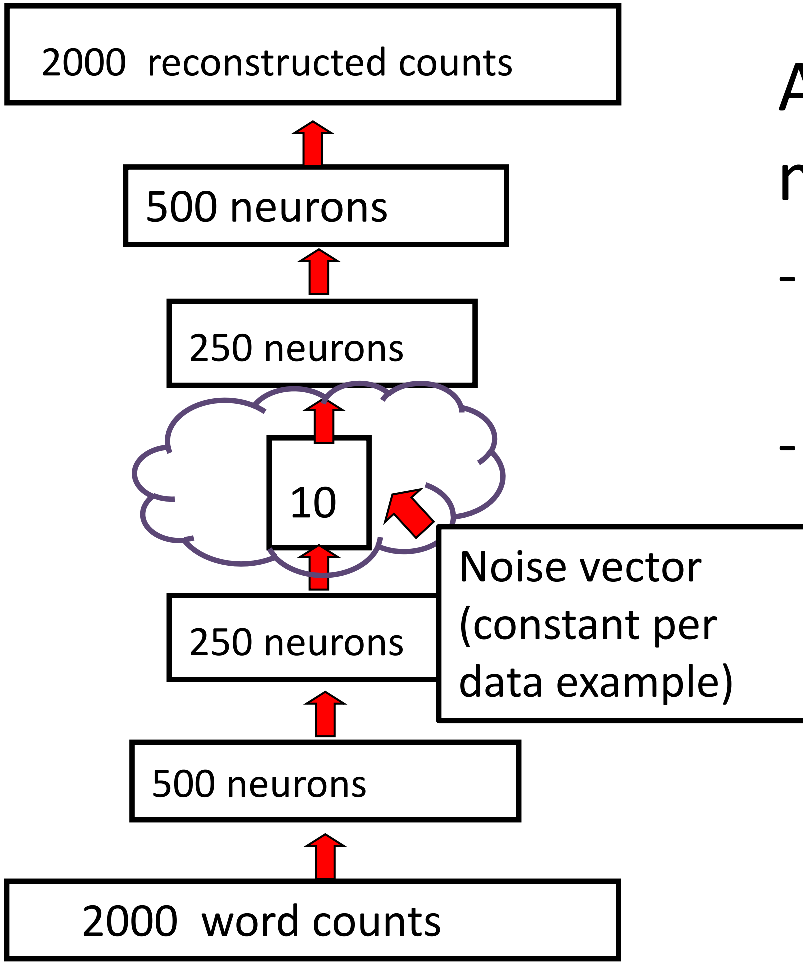
# Clusters




Images from Hinton, 2007

# Search




Add noise to input to middle layer

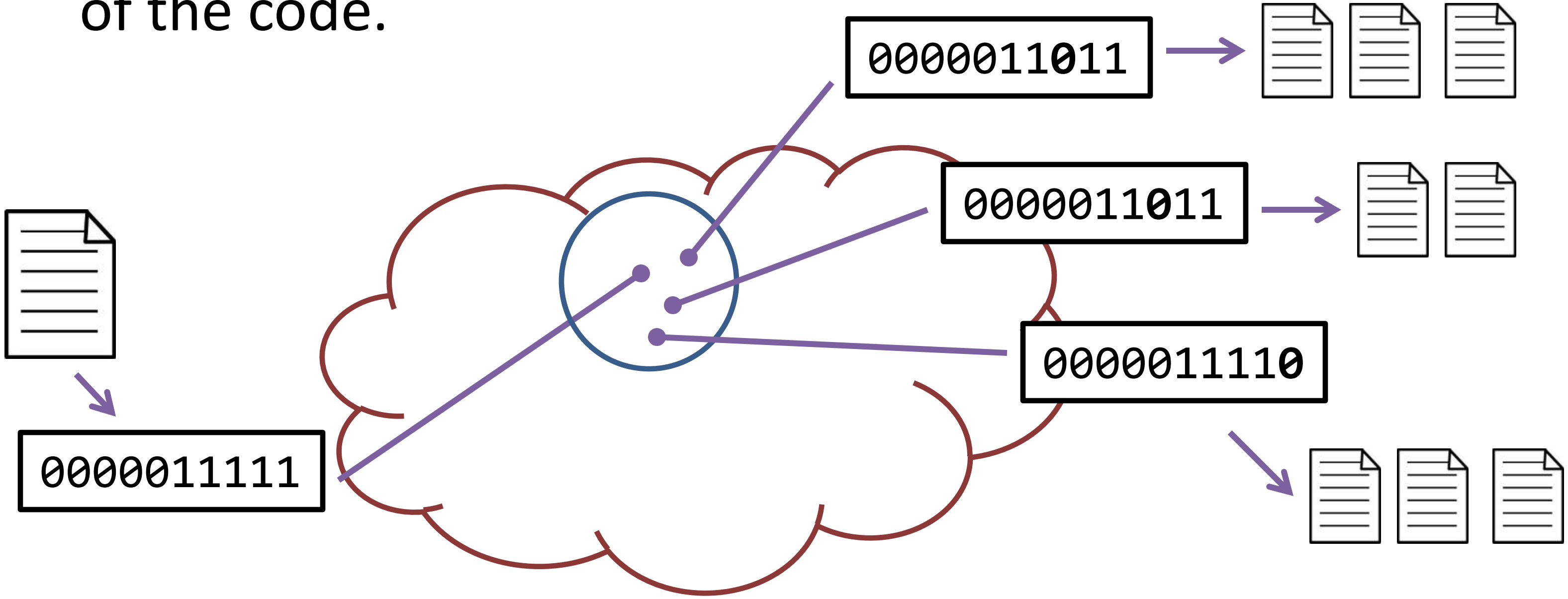
- Forces output to become bimodal
- Round values to 0 or 1 to form a binary vector (ie, code)

Hinton, 2007

# Search

Use the binary codes as a key/hash documents

To find a similar document, calculate binary code and then retrieve documents that correspond to small deviations of the code.




Salakhutdinov and Hinton, 2007

# References

Hinton, G. *2007 NIPS Tutorial on: Deep Belief Nets* [PowerPoint slides]. Retrieved from http://www.cs.utoronto.ca /~hinton/.

Hinton, G. *UCL Tutorial on: Deep Belief Nets* [PowerPoint slides]. Retrieved from http://www.cs.utoronto.ca /~hinton/.

Hinton, G., Osindero, S. and The, Y. (2006) A fast learning algorithm for deep belief nets. *Neural Computation*, **18**, pp 1527-1554.

Hinton, G. and Salakhutdinov, R. (2006) Reducing the dimensionality of data with neural networks. *Science*, **313**:5786, pp. 504 – 507.

Salakhutdinov, R and Hinton, G. (2007) Semantic Hashing. *Proceedings of the SIGIR Workshop on Information Retrieval and applications of Graphical Models*, Amsterdam.

*Restricted Boltzmann Machines (RBM).* Retrieved from http://deeplearning.net/ tutorial/rbm.html#rbm

Carreira-Perpinan, M and Hinton, G. (2005) On Contrastive Divergence Learning. *Artificial Intelligence and Statistics*, Barbados.